Amendments to the Specification:

Please amend the third paragraph beginning on page 12 of the current specification as follows:

The present application is directed to a host-based IDS on an HP-UX intrusion detection system that enhances local host-level security within the network. It should be understood that the present invention is also usable on, for example, ~~Eglinux, solaris, aix~~ LINUX, SOLARIS, AIX, and ~~windows~~ WINDOWS 2000 operating systems. It does this by automatically monitoring each configured host system within the network for possible signs of unwanted and potentially damaging intrusions. If successful, such intrusions could lead to the loss of availability of key systems or could compromise system integrity.

Please amend the sixth paragraph beginning on page 17 of the current specification as follows:

A secure communications link. The host-based IDS needs a means of stopping an attacker from observing the traffic between its components and possibly sending false data to disrupt its operations. An encrypted link can prevent this from happening.

Please amend the ninth paragraph beginning on page 17 of the current specification as follows:

The host-based IDS 50 analyzes this information against stored configured attack scenarios. The host-based IDS 50 then identifies possible intrusions and misuse immediately following any suspected activity and simultaneously communicates an alert and detailed information on the potential attack to the host-based IDS GUI.

Please amend the sixth paragraph beginning on page 19 of the current specification as follows:

Figure 2 is a diagram illustrating a logical architecture of an intrusion detection system according to the principles of the present invention. Figure 2 provides greater detail than Figure 1 regarding agent 60.

Please amend the fifth paragraph beginning on page 20 of the current specification as follows:

The idsagent 210 creates the low and high bandwidth connections between itself and the agent processes. The low bandwidth connections are built using POSIX message queues. The high bandwidth connection is built using a memory-mapped (mmap) file. The advantage of the memory mapped file is that it does not require a system call to read or write data from/to it. Processes access the mmap file via a pointer in their address region.

Please amend the fourth paragraph beginning on page 21 of the current specification as follows:

2. It will execute the alert response scripts and pass ~~then~~ them the alert details as command line arguments. The alert response script are located in /opt/ids/lbin/ids_alertResponse

Please amend the fifth paragraph beginning on page 21 of the current specification as follows:

3. It will package the alert text, encrypt it, and write it to the GUI for display.

Please amend the second paragraph beginning on page 22 of the current specification as follows:

The ECS engine in the ids is embedded within the IDS correlator process ~~230~~ 220 to improve performance. The rate of data generated from the kernel is very high; the path from the kernel to the correlator must be as short as possible. The ECS engine in host-based IDS 50 has been built to parse and understand kernel audit records, system log files and other data sources. It uses a meta-description language (MDL) to define what a record in a data stream looks like. The MDL specifications allow for fast parsing of the event streams.

Please amend the third paragraph beginning on page 24 of the current specification as follows:

The cron daemon 250 reads the crontab[[l]] for user "ids" and executes the idssc program at the specified intervals. This is used to start and stop surveillance groups.

Please amend the fourth paragraph beginning on page 24 of the current specification as follows:

Once an intrusion is detected, the idsagent [[200]] <u>210</u> will execute the binary located at the path /opt/ids/ Response. Each command line argument to this binary will be a field of the alert.

Please amend the seventh paragraph beginning on page 24 of the current specification as follows:

The idscor 220 process contains the correlator which processes the raw events and determines if an intrusion has occurred. It obtains the data from the memory mapped file created by the idsagent [[200]] <u>210</u>. The idscor 220 is linked via this mmap file to the idssysdsp 230 and the idskerndsp 240. The idscor 220 will poll each file in turn looking for data.

Please amend the third paragraph beginning on page 25 of the current specification as follows:

The idscor 220 is forked by the idsagent [[200]] <u>210</u> when a surveillance group is being started. It is passed on the command line the id's of the low-bandwidth status and command channels, and the high-bandwidth memory-mapped file channel. The idscor 220 then receives commands over the command channel from the idsagent [[200]] <u>210</u>.

Please amend the sixth paragraph beginning on page 25 of the current specification as follows:

The idssysdsp 230 is started by the idsagent [[200]] 210 when a surveillance group is started. It is passed on the command line the name of the memory-mapped file to communicate with the idscor 220.

Please amend the first paragraph beginning on page 26 of the current specification as follows:

The idskerndsp 240 is started by the idsagent [[200]] 210 when a surveillance group is started. The idskerndsp 240 is passed on the command line the name of the memory-mapped file to communicate with the idscor 220.

Please amend the second paragraph beginning on page 26 of the current specification as follows:

The idskerndsp 240 is also responsible for enabling audit of selected audit records according to which templates are being used as described in greater detail below. When the idsagent [[200]] 210 is starting the surveillance group, it will send data to the idskerndsp 240 indicating which audit records the detection templates require. The idskerndsp 240 will then do a ioctl call to the idds driver to enable those system calls for audit.

Please amend the sixth paragraph beginning on page 26 of the current specification as follows:

In blocking mode, the IDDS subsystem 270 will wait until there is space in the buffer for the new audit record. The calling process is blocked until space becomes available, for the audit record. Once space is made available the audit record is stored in the buffer and the process continues. Blocking mode sacrifices some system performance for security.

Please amend the first paragraph beginning on page 27 of the current specification as follows:

In non-blocking mode, the host-based IDS will discard the audit record information if no

space is available in the buffer. Non-blocking mode sacrifices security for system performance.

Please amend the third paragraph beginning on page 27 of the current specification as follows:

The developers of the host-based IDS have identified a subset of system calls on HP-UX which have security relevance, and occur ~~frequency~~ frequently in exploits. The IDDS subsystem 270 in the kernel will record these system calls as they occur. The system call trace is made available to the host-based IDS agents via a device driver in the kernel: /dev/idds. The host-based IDS agent opens this device and reads system call information as it occurs.

Please amend the ninth paragraph beginning on page 27 of the current specification as follows:

- Can ~~operating~~ operate in "blocking" mode: system calls are halted until the host-based IDS agent can catch up reading the data stream.

Please amend the second paragraph beginning on page 30 of the current specification as follows:

Refer now to Figure 3, where a flow diagram ~~illustrating~~ illustrates an example of how intrusions are detected. At step 305, a user process makes a libc library call: the open() or unlink() calls for example. At step 310, the libc library translates the call into a system call and calls the system call dispatch entry point. At step 315, the initial component of the syscall handler checks to see if this system call is being audited by the host-based IDS. If the system call is being audited, the initial component of the syscall handler gathers some header related information: user id, group id, timestamps, process id, etc. At step 315 as the system call is processed, information is stored in temporary buffers. This information corresponds to the arguments of the system call and any further data that is reported. Once the system call completes, the return value and errno value are recorded. At that point the entire record is placed in a circular buffer in the kernel audit driver (step 320). At step 325, the system call returns to the user process context. At step 330, a read() of /dev/idds has forced the IDDS kernel driver [[370]] 270 to read the next audit record block from the circular buffer. At step 335, the system

call block is passed up to the user context of the idskerndsp 240 in response to the read () call. At step 340, the idskerndsp 240 reformats the raw binary audit record as ASCII data in a format that the correlator idscor 220 will understand. At step 345, the correlator idscor receives the data and parses it using MDL into an internal ~~vent~~ event format. At step 350, the detection templates take this internal event format and process it. The ECS using a detection template may decide that an intrusion has occurred. At step 355, the detection template generates an alert message in the internal event format. The idscor 220 takes this alert message and reformats it as an ASCII message. This message text is sent on the status output channel to the idsagent 210. At step 360, the idsagent 210 is polling the status connection from the idscor 220 periodically. The IDS agent [[200]] 210 receives the alert message and reads it from the status connection. At step 365, idsagent 210 then executes any local alert response script and passes them the alert details. At step 370, the alert is logged to the local alert log file. At step 375, the idsagent 210 reformats the alert for the GUI. At step 380, the alert message is sent to the idsSSLagent 200.

Please amend the second paragraph beginning on page 31 of the current specification as follows:

Figure 4 is an illustration of a logical architecture, similar to Figure 2, but showing in greater detail the idscor 220 which is shown in Figure 4 as correlator 1, correlator 2 . . . . correlator n. Information flows upward from data source process 1, data source process 2, data source process 3 . . . data source process n to any or all of the correlators 1-n. It should be noted that data process source 1 corresponds to idssysdsp 230 in Figure 2 and data source process 2 corresponds to the idskerndsp 240. Further, it should be appreciated that the host-based IDS 50 may not necessarily have all the[[se]] processes that the overall architecture of the host-based IDS 50 supports. In fact, the host-based IDS 50 can support multiple correlators and multiple data sources. However, there does not need to be a corresponding number of data source processes and correlators. In other words, there can be one or more correlators with many data source processes or one or more data source processes with only a single correlator. Each data source process and correlator should add some measure of intrusion detection capability [[or]] for the expense of processing speed and additional processing resources required. The IDS monitor process 410 is the main control process and corresponds to a combination of the functionality of the idSSLAgent 200 and the idsagent 210 described with respect to Figure 2.

The IDS monitor 410 is responsible for connecting with all the functional components depicted in Figure 4 and it is responsible for taking commands issued by the user and translating them into commands to send on to the processes that are running below it. In addition, the IDS monitor 410 is responsible for monitoring the status of the processes running on the system and it is responsible for gathering alert information generated by the correlators 1-n and forwarding that to the GUI 55. The other task the IDS monitor process 410 must perform is if the user has scheduled to run surveillance schedules at a future time, the IDS monitor 410 is responsible for initiating the processing. The IDS monitor 410 is responsible for executing with the response scripts 260 (shown in Figure 4 as ~~C-Magent~~ C-M Agent 260). When an alert is detected, the alert will be written in a notification log (shown as local alert file in Figure 2). A configuration file 440 details how the host-based IDS 50 is put together, what circuits are installed, etc. The IDS monitor 410 interacts with cron job 250 as described with respect to Figure 2 to launch surveillance schedules at a specific time.

Please amend the second paragraph beginning on page 32 of the current specification as follows:

The correlator 1, correlator 2, . . . correlator is a layer which processes the data coming off the system in conjunction with the templates to determine if there has been an intrusion. As depicted in Figures 1 and 2, there is only one correlator present, but the architecture supports multiple correlators (Figure 4).

Please amend the third paragraph beginning on page 32 of the current specification as follows:

Correlator 1 uses the previously described ECS technology. Correlator 2-n can use other correlation technologies. Communication occurs between the correlators 1-n and data source using memory mapped files 1-n and processes 1-n. The memory mapped files ~~are~~ provide a low overhead, high bandwidth connection between processes running on a system. Specifically by generating data and pulling data into a memory mapped file by reading the data, the host-based IDS 50 does not generate system calls. These memory mapped files 1-n are created by the IDS monitor 50 when the correlators 1-n are being started. When the IDS monitor 50 starts the correlator it also creates a connection to send commands from the IDS monitor 410 to the

correlators (see arrows 445) and creates other connections (446, 448, 450) from the correlators to the IDS monitor 410.

Please amend the fourth paragraph beginning on page 32 of the current specification as follows:

The IDS monitor 410 also interacts with a management system 460. The management system 460 includes an IDS security administration component 465 and an enterprise management component 470. The IDS monitor 410 sends notifications to the enterprise management component 470. The IDS monitor 410 interacts bi-directionally with the IDS security administration component 465 including configuration, notification, control and status. The IDS security administration component 465 has a GUI [[4]]55 for displaying alert notifications. The enterprise management component 470 provides the application launch and node list to the IDS security administration component 465. The IDS security administration component 465 also sends an alert configuration to the enterprise management component 470.

Please amend the second paragraph beginning on page 33 of the current specification as follows:

Referring now to Figure 5, Figure 5 illustrates a more detailed view of the IDS security administration component 465 (Figure 4). It should be noted that one administration component 465 can control many IDS agent nodes.

Please amend the third paragraph beginning on page 33 of the current specification as follows:

The IDS security administration component 465 is responsible for creating surveillance schedules and groups and communicating these with the respective IDS host-based agent nodes. A software install/update module 505 which is located in the IDS GUI 55 can be used to install or update software on IDS agent nodes as updates become available. A security configuration preferences module 510 allows the administrator to save various security configuration preferences for the particular GUI. Also, the security schedule groups and configurations can be saved using module 510. Operation module 515 is used for query, shutdowns and various other

operations of the GUI and IDS agent nodes. The IDS browser 520 is used to sort, query and search alerts. An IDS enterprise interface module 550 allows the IDS GUI 55 to be plugged into various other enterprise architectures such as HP OpenView VP/0 architecture. Thus, the IDS GUI 55 can be managed from HP OpenView VP/0 management software. When a surveillance schedule is generated the program object 565 is generated. When the preferences are saved, then the preference object 570 is generated. The program object and preference object send information to the IDS administrative core 580 which in turn communicates control configuration and notification status along secure connections to the respective IDS agent nodes. A node list 585 is generated by the operation module 515. The object node list 585 is generated when the processes are stopped, started and queried the status of the agent nodes which in turn is packaged as an object file and sent to the IDS administrative core 580 and then is forwarded on to the respective IDS agent node. The IDS administrative core 580 is responsible for secure communications with the multiple IDS agent nodes.

Please amend the third paragraph beginning on page 34 of the current specification as follows:

The host-based IDS product includes a number of detection templates which have been created and pre-configured. When the user initially selects the Surveillance Group menu item, the Select a Surveillance Group box will open. The predefined detection templates will become visible when the user either presses the Edit button in this box to modify an existing surveillance group or the New button to create a new surveillance group.

Please amend the fifteenth paragraph beginning on page 62 of the current specification as follows:

A virus ~~Virus~~ exhibits many of the same characteristics of a host-based attack and so the IDSSO according to the present invention provides a second tier virus protection. Virus protection software operates by searching (in one form or another) for known virus codes within system, application, or data files, or in data coming in via the network. Sometimes the viral code sequences can be removed (the file is "repaired"), other times an alert is provided and activity related to that file is blocked. Such virus protection schemes are completely ineffective against

new and unknown virus codes, or against viruses whose codes have not been added to the published list of known viruses [[.]] and against "polymorphic" viruses that alter their own codes as they propagate.

Please remove the sixteenth paragraph beginning on page 62 of the current specification as following:

~~A host-based IDS can also provide second tier virus protection. All current virus protection software operates by searching in one form or another for known virus codes within system, application, or data files or in data coming in via the network. Sometimes the viral code sequences can be removed (the file is "repaired"), other times an alert is provided and activity related to that file is blocked. Such virus protection schemes are completely ineffective against new and unknown virus codes, against viruses whose codes have not been added to the published library of known viruses and against "polymorphic" viruses that alter their own codes as they propagate.~~

Please amend the fourth paragraph beginning on page 63 of the current specification as follows:

The templates (circuits) in the correlator will generate an ASCII text message if an intrusion is detected. The event must be sent from the correlator 220 to the outside world, which is the idsagent [[200]] 210. Remember, the correlator process idscor 230 is executing as a single thread, so the only way to asynchronously read an output event from the engine core is to define a callback function. The callback function is called by the engine core whenever it wishes to send an event to the outside world.

Please amend the second paragraph beginning on page 64 of the current specification as follows:

The engine core needs to load an MDL file which specifies the format and layout of all events that the engine must deal with. The mdlFile global is set from the environment variables created when the idscor process [[130]] 220 is started. It specifies the location of the mdl file, which should be in /opt/ids/lib/mdl.md by default. The ECS_MDL_MD environment variable is

created for further use by the engine. However, the ECS engine does not load the MDL file directly. Instead, it loads a file which contains a line that specifies the full path to the shared libraries for all the endecoders it must load.

Please amend the second paragraph beginning on page 65 of the current specification as follows:

The correlator loads and unloads templates (circuits) in response to commands from the idsagent process [[110]] 210.

Please amend the second paragraph beginning on page 65 of the current specification as follows:

A group of templates is loaded which form a surveillance group defined in the GUI. The command from the idsagent [[110]] 210 is in the form of an ASCII string, with the parameters to the command separated by spaces. The comment block on this function is detailed enough to explain what it does. If one of the templates fails to load, then all of the templates are unloaded. Each circuit to be loaded is specified as a name in the parameter list. The idscor [[130]] 220 will load the circuit using the circuit name. For simplicity, the idscor [[130]] 220 assumes that each circuit to be loaded has an associated data and fact store. So the sequence of steps to load a template group are (for each circuit specified in the group):